Parallel Adaptive and Time-stabilizing Schemes for Constant-Coefficient Parabolic PDEs *

appeared in Computers & Mathematics with applications,Vol. 24, No. 10, pp. 33-53, 1992

Dganit Amitai †
Amir Averbuch †
Moshe Israeli ‡ †
Samuel Itzikowitz †‡

† Department of Computer Science
School of Mathematical Sciences
Tel Aviv University, Tel Aviv 69978, Israel
‡ Faculty of Computer Science
Technion
Haifa 32000, Israel

Abstract

New asynchronous and time stabilizing finite-difference methods for constant coefficient parabolic PDEs are presented. These schemes remove the synchronization overhead and are suitable for shared-memory, message-passing, single- or multi-user multiprocessors. They generalize previous schemes by allowing adaptive and non-constant time increments. They are characterized by the *time-stabilizing* property: the derivatives of their time-level with respect to the spatial coordinates are bounded as the number of time-steps increases, and they vanish as the spatial mesh-size approaches zero. Therefore, unlike others, our schemes are suitable for multiprocessors with non-identical processors or unbalanced workload, provided that the differences among their speeds/loads are not too large.

The consistency, stability, convergence and the time-stabilizing property of our methods are analyzed in detail. When implemented on the shared-memory multi-user Sequent Balance machine, they show to provide an excellent efficiency and in certain

*A reduced version of the paper was presented at the Fifth SIAM Conference on Parallel Processing for Scientific Computing, March 24-27, 1991, Houston, USA.

†This research was supported by the fund for promotion of research at the technion and by the technion V.P.R. - E.& J. Bishop research fund.

‡The work of this author was supported by research grant #337 of the Israeli National Council for Research and Development.

cases it is tripled, compared to other existing schemes. However, their truncation error is of lower order ($O(\Delta x_l)$ instead of $O(\Delta x_l^2)$). Hence, because of the dimensional additivity of the truncation error, these schemes are less accurate especially for multidimensional problems. For such problems, if high accuracy is sought, in general one should use higher order schemes. Nevertheless, under certain circumstances, when the spatial increments are relatively large, the performance of our asynchronous schemes (i.e both their efficiency and accuracy) may be about the same as compared to their synchronous counterparts, and sometimes even better.

Key-Words:
Parallel asynchronous and synchronous finite difference methods, finite difference schemes with adaptive time increments, parabolic finite difference approximations with constant coefficients, synchronization overhead, time-stabilizing finite difference methods.

1 Introduction

The introduction of parallel computer architectures has challenged the existing serial algorithms for the numerical solution of PDEs. Extensive work has been done on updating existing methods to fit these new architectures (see [6],[8]). Yet, most of this work assumes some correlation between the processors, achieved by synchronization mechanisms. For example, numerical iterative solutions to PDEs are usually based on the assumption that the time-steps are *synchronized.* This is equivalent to assuming that the algorithm is governed by a global clock so that the start of each time-step is simultaneous for all processors. The overhead associated with such synchronization, limits the speedup expected from a parallel machine. Obviously, an asynchronous algorithm removes the synchronization penalty since each processor can execute more time-steps without the need to wait for certain results of other processors.

The use of *asynchronous iterations* for iterative solutions of systems of linear equations is discussed in [5]. Asynchronous Iterative Methods for Multiprocessors are also discussed in [4], and are used for the solution of ordinary differential equations by [7].

In [2] and [3] asynchronous and corrected-asynchronous finite-difference schemes were developed for the multidimensional heat equation. They are based on constant time-increments. The purpose of this paper is to generalize these schemes and to develop asynchronous methods which are based on non-constant time increments as to improve their efficiency substantially.

In Sec. 2 we introduce our model problem and corresponding finite difference schemes including our new scheme. Consistency, stability, convergence and the time-stabilizing property of our scheme are investigated in Sec. 3. The possibility of obtaining other similar time-stabilizing approximations is addressed in Sec. 4. Results of numerical experiments are presented in Sec. 5. Finally, Sec. 6 summarizes the principal results of our paper.

2 The Model Problem and its Finite Difference Schemes

We demonstrate our proposed method on the multidimensional heat equation:

$$\frac{\partial u}{\partial t} = \sum_{l=1}^{d} a_l^2 \frac{\partial^2 u}{\partial {x_l}^2} + F(\vec{x}, t) \tag{1}$$

with appropriate initial and boundary conditions.

We will consider here only the constant coefficients case. The non-constant coefficients case will be examined elsewhere.
For example, in the one-dimensional case we have

$$\frac{\partial u}{\partial t} = a^2 \frac{\partial^2 u}{\partial x^2} + F(x,t) \qquad \text{for} \quad 0 < x < 1 \quad 0 < t < T$$

with the initial condition,

$$u(x,0) = f(x) \qquad 0 \le x \le 1$$

and the Dirichlet boundary conditions,

$$u(0,t) = g_1(t) \qquad 0 < t \le T$$

$$u(1,t) = g_2(t) \qquad 0 < t \le T$$

or else Neumann conditions can be specified.

In order to motivate and clarify our new *Time-Stabilizing* (TS) scheme, we start below with the well known synchronous Euler scheme (denoted by SY), derive the asynchronous (AS) and the corrected asynchronous (CA) schemes of [2] and [3] and finally introduce our new TS scheme. By approximating Eq. (1) at the grid point $(\vec{x}_{\mathbf{j}}, t_n)$ where $\mathbf{j} = (j_1, j_2, \ldots, j_d)$ and $t_n = n\Delta t$, with the forward Euler synchronous (lock-step) finite difference scheme, we obtain our SY scheme:

$$u_{\mathbf{j}}^{n+1} = u_{\mathbf{j}}^{n} + \sum_{l=1}^{d} r_l \delta_l^2 u_{\mathbf{j}}^{n} + (\Delta t) F_{\mathbf{j}}^{n} \qquad r_l = a_l^2 \frac{\Delta t}{(\Delta x_l)^2} \tag{2}$$

In Eq. (2), $u_{\mathbf{j}}^n$ and $F_{\mathbf{j}}^n$ denote the values of u and F at grid point $(\vec{x}_{\mathbf{j}}, t_n)$ respectively, $u_{\mathbf{j}}^{n+1}$ is the value of u at $(\vec{x}_{\mathbf{j}}, t_{n+1})$, and δ_l^2 denotes the central difference in the space coordinate l. Thus,

$$\delta_l^2 u_{\mathbf{j}}^n = u_{\mathbf{j}-\mathbf{1}_l}^n - 2u_{\mathbf{j}}^n + u_{\mathbf{j}+\mathbf{1}_l}^n$$

where $\mathbf{1}_l = (0, 0, \ldots, \underbrace{1}_{l-th\ component}, \ldots, 0, 0)$ while $\mathbf{j} - \mathbf{1}_l$ and $\mathbf{j} + \mathbf{1}_l$ are the neighbors of the point $\mathbf{j}$ in the positive and the negative directions along the l-axis, respectively.

Note that unlike the following approximations, the terms of the right hand side of Eq. (2) are all evaluated at the *same* time level $t = n\Delta t$.

In the synchronous scheme of Eq. (2), the $(n+1)^{th}$ time-step of the grid point $\mathbf{j}$ is being performed only after the n^{th} time-step of all other grid points had been completed. We now remove this synchronization restriction (and yet maintaining Eq. (2)), allowing the time-steps of any grid point $\mathbf{j}$ to be performed asynchronously by using the most recent available

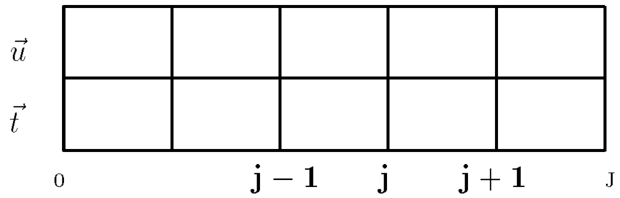


Figure 1: Solution array $\vec{u}$ and time array $\vec{t}$ for the one-dimensional case

values at the neighboring grid points $\mathbf{j} \pm \mathbf{1}_l$ $(1 \leq l \leq d)$. Taking a **local** view of the behavior at the grid point $\mathbf{j}$ and assuming a constant time interval Δt at all grid points, its time level after completing $n_{\mathbf{j}}$ time-steps is $n_{\mathbf{j}} \Delta t$ (note that there is no global time step n, since there is no global synchronization). We denote the time-level difference of the grid point $\vec{\mathbf{j}}$ relative to its neighbors in the negative and positive l-direction, by $\alpha_{\mathbf{j}_l}^{n_{\mathbf{j}}}$ and $\beta_{\mathbf{j}_l}^{n_{\mathbf{j}}}$, respectively. Then we get the following asynchronous scheme of [2] and [3] which is denoted here by AS:

$$u_{\mathbf{j}}^{n_{\mathbf{j}}+1} = u_{\mathbf{j}}^{n_{\mathbf{j}}} + \sum_{l=1}^{d} r_l \left(u_{\mathbf{j}-\mathbf{1}_l}^{n_{\mathbf{j}}+\alpha_{\mathbf{j}_l}^{n_{\mathbf{j}}}} - 2u_{\mathbf{j}}^{n_{\mathbf{j}}} + u_{\mathbf{j}+\mathbf{1}_l}^{n_{\mathbf{j}}+\beta_{\mathbf{j}_l}^{n_{\mathbf{j}}}}\right) + (\Delta t) F_{\mathbf{j}}^{n_{\mathbf{j}}}, \qquad r_l = a_l^2 \frac{\Delta t}{(\Delta x_l)^2} \qquad (3)$$

In order to clarify our later discussions, we now introduce into Eq. (3) a slightly different notation. We consider two arrays of data, each with as many components as the number of grid points, (see Fig. 1). One array, denoted by $\vec{u}$, represents an array of the most recent solution values of the scheme as calculated by the processors. The other array, denoted by $\vec{t}$, contains the corresponding most recent time-levels as calculated for each grid point. A processor is allowed to change only its own values, and to read its neighbors' values. Therefore, the arrays are global in the sense that access is allowed and information is shared by all processors. In the shared memory machine model we have two global arrays whose pairs of items $u_{\mathbf{j}}$ and $t_{\mathbf{j}}$ are *atomized* (encapsulated) in such a way that the values must be accessed (read/write) together. In a message-passing type of architecture, we need the same mechanism of *atomizing* $\vec{u}$ and $\vec{t}$. However, in this case relevant array-components will reside in the local memories of each processor. In any case, a time-step counter, denoted by $n_{\mathbf{j}}$, is maintained for each grid point. Its value is incremented each time the processor had completed a new iteration (time-step) of this grid point.

Let $t_{\mathbf{j}\pm\mathbf{1}_l}^{n_{\mathbf{j}}}$ denote the most recent available time-values of the neighbors in the positive and negative l-direction, read by the processor evaluating u-value at the grid point $\mathbf{j}$ when its counter value was showing $n_{\mathbf{j}}$. Hence, if, for example, the neighbor of the grid point $\mathbf{j}$ in the positive l-direction is continuously updated faster than $\mathbf{j}$ itself, then $t_{\mathbf{j}+\mathbf{1}_l}^{n_{\mathbf{j}}} > t_{\mathbf{j}}^{n_{\mathbf{j}}}$. If it is continuously updated slower than $\mathbf{j}$, then $t_{\mathbf{j}+\mathbf{1}_l}^{n_{\mathbf{j}}} < t_{\mathbf{j}}^{n_{\mathbf{j}}}$.

We now write Eq. (3) in the form

$$u_{\mathbf{j}}(t_{\mathbf{j}}^{n_{\mathbf{j}}+1}) = u_{\mathbf{j}}(t_{\mathbf{j}}^{n_{\mathbf{j}}}) + \sum_{l=1}^{d} r_l \bar{\delta}_l^2 u_{\mathbf{j}}^{n_{\mathbf{j}}} + (\Delta t) F_{\mathbf{j}}^{n_{\mathbf{j}}} \quad , \quad r_l = \frac{a_l^2 \Delta t}{(\Delta x_l)^2}. \qquad (4)$$

In Eq. (4), $\bar{\delta}_l^2$ denotes a central difference in the direction of the space coordinate l, corresponding to the grid point $\mathbf{j}$, with the u-values of the neighbor points in the positive and negative l-direction evaluated not at the time-level $t_{\mathbf{j}}^{n_{\mathbf{j}}}$ of $\mathbf{j}$ but at their *own* most recent time levels: $t_{\mathbf{j+1}_l}^{n_{\mathbf{j}}}$ and $t_{\mathbf{j-1}_l}^{n_{\mathbf{j}}}$, respectively. Thus,

$$\bar{\delta}_l^2 u_{\mathbf{j}}^{n_{\mathbf{j}}} = u_{\mathbf{j-1}_l}(t_{\mathbf{j-1}_l}^{n_{\mathbf{j}}}) - 2u_{\mathbf{j}}(t_{\mathbf{j}}^{n_{\mathbf{j}}}) + u_{\mathbf{j+1}_l}(t_{\mathbf{j+1}_l}^{n_{\mathbf{j}}})$$

In the one-dimensional case the asynchronous scheme (AS) becomes,

$$\begin{aligned} u_j(t_j^{n_j+1}) &= u_j(t_j^{n_j}) + r\bar{\delta}_s^2 u_j^{n_j} + (\Delta t)F_j^{n_j} = \\ &= ru_{j-1}(t_{j-1}^{n_j}) + (1-2r)u_j(t_j^{n_j}) + ru_{j+1}(t_{j+1}^{n_j}) + (\Delta t)F_j(t_j^{n_j}) \end{aligned}$$

However, as shown in [2] and [3] this scheme is no longer consistent with Eq. (1). Yet, it can be modified to produce a consistent scheme by using the time-level values at the neighboring grid points. We thereby obtain the Corrected-Asynchronous (CA) scheme of [2] and [3] as follows. For the calculation of the updated solution $u_{\mathbf{j}}^{n_{\mathbf{j}}+1}$, the CA scheme requires the components' values of $\vec{u}$ as before, but also the components' values of $\vec{t}$ corresponding to the grid point $\mathbf{j}$ and to the relevant neighboring points. With our notations, the finite difference CA scheme becomes:

$$u_{\mathbf{j}}(t_{\mathbf{j}}^{n_{\mathbf{j}}+1}) = u_{\mathbf{j}}(t_{\mathbf{j}}^{n_{\mathbf{j}}}) + \frac{\Delta t}{K}\sum_{l=1}^{d}\frac{a_l^2\bar{\delta}_l^2 u_{\mathbf{j}}^{n_{\mathbf{j}}}}{(\Delta x_l)^2} + \frac{\Delta t}{K}F_{\mathbf{j}}(t_{\mathbf{j}}^{n_{\mathbf{j}}}) \tag{5a}$$

where,

$$K = 1 + \sum_{l=1}^{d}\frac{a_l^2\delta_l^2 t_{\mathbf{j}}^{n_{\mathbf{j}}}}{(\Delta x_l)^2} \tag{5b}$$

and

$$\delta_l^2 t_{\mathbf{j}}^{n_{\mathbf{j}}} = t_{\mathbf{j+1}_l}^{n_{\mathbf{j}}} - 2t_{\mathbf{j}}^{n_{\mathbf{j}}} + t_{\mathbf{j-1}_l}^{n_{\mathbf{j}}} \tag{5c}$$

In order to simplify our analysis to follow, we will use the following short notations:

$$t \triangleq t_{\mathbf{j}}^{n_{\mathbf{j}}}, \quad t^+ \triangleq t_{\mathbf{j}}^{n_{\mathbf{j}}+1}, \quad t_{\pm l} \triangleq t_{\mathbf{j\pm 1}_l}^{n_{\mathbf{j}}}, \quad n \triangleq n_{\mathbf{j}}, \quad n_{\pm l} \triangleq n_{\mathbf{j\pm 1}_l}$$

where $n_{\mathbf{j\pm 1}_l}$ are the most recent counters' values of the neighbors in the positive and negative l-direction, examined when the local counter of grid point $\mathbf{j}$ was $n_{\mathbf{j}}$,

$$u \triangleq u_{\mathbf{j}}(t), \quad \frac{\partial u}{\partial t} \triangleq (\frac{\partial u}{\partial t})_{\mathbf{j}}(t), \quad u^+ \triangleq u_{\mathbf{j}}(t^+),$$

$$\frac{\partial u}{\partial x_l} \triangleq (\frac{\partial u}{\partial x_l})_{\mathbf{j}}(t), \quad \frac{\partial^2 u}{\partial t \partial x_l} \triangleq (\frac{\partial^2 u}{\partial t \partial x_l})_{\mathbf{j}}(t), \quad u_{\pm l} \triangleq u_{\mathbf{j\pm 1}_l}(t), \text{ and } \bar{u}_{\pm l} \triangleq u_{\mathbf{j\pm 1}_l}(t_{\pm l}).$$

Note that while $\bar{u}_{\pm l}$ denote the neighbors' values at their *own* time-levels when the counter of the grid point $\mathbf{j}$ is showing $n_{\mathbf{j}}$, $u_{\pm l}$ denote their (estimated) values at the same time-level as of the grid point $\mathbf{j}$ (i.e. $t_{\mathbf{j}}^{n_{\mathbf{j}}}$). With these notations we rewrite Eq. (5a) in the following form:

$$u^{+} = u + \frac{\Delta t}{K}\sum_{l=1}^{d}\frac{a_l^2\bar{\delta}_l^2 u}{(\Delta x_l)^2} + \frac{\Delta t}{K}F \tag{6}$$

In one dimension it becomes:

$$u^{+} = u + \frac{a^2\Delta t}{(\Delta x)^2 + a^2\delta^2 t}\bar{\delta}^2 u + \frac{(\Delta t)(\Delta x)^2}{(\Delta x)^2 + a^2\delta^2 t}F$$

We now generalize Eq. (6) and unlike [2] and [3] we allow Δt to vary and to depend upon the grid point $\mathbf{j}$ and upon the current time-step counter-value of that point, $n_{\mathbf{j}}$. With the above notations our new scheme becomes:

$$u^{+} = u + \frac{\Delta t_{\mathbf{j}}^{n_{\mathbf{j}}}}{K}\sum_{l=1}^{d}\frac{a_l^2\bar{\delta}_l^2 u}{(\Delta x_l)^2} + \frac{\Delta t_{\mathbf{j}}^{n_{\mathbf{j}}}}{K}F \tag{7}$$

where K is given by Eqs. (5b) and (5c). Equation (7), along with the appropriate expression for $\Delta t_{\mathbf{j}}^{n_{\mathbf{j}}}$ as determined by the stability restriction (lemma 2) is our new *Time-Stabilizing* (TS) scheme.

In one dimension we obtain

$$u^{+} = u + \frac{a^2\Delta t_j^{n_j}}{(\Delta x)^2 + a^2\delta^2 t}\bar{\delta}^2 u + \frac{(\Delta t_j^{n_j})(\Delta x)^2}{(\Delta x)^2 + a^2\delta^2 t}F$$

As discussed in [3], CA scheme can be applied only when the differences between the time levels of a grid point and of its relevant neighbors (i.e $\alpha_{\mathbf{j}}^{n_{\mathbf{j}}}$ and $\beta_{\mathbf{j}}^{n_{\mathbf{j}}}$ of Eq. (3)) are bounded. Therefore, the CA algorithm cannot be implemented when, for example, one or more processors are slower than the others, since then $\alpha_{\mathbf{j}}^{n_{\mathbf{j}}}$ and $\beta_{\mathbf{j}}^{n_{\mathbf{j}}} \to \infty$ as $n_{\mathbf{j}} \to \infty$ so that the gap between the time-levels becomes unbounded. We call this the "gapping" effect. In addition, a stability condition of the CA scheme, which has to be examined before each time-step, reduces its efficiency substantially.

However, as shall be described later, stability of the TS algorithm dictates the time increment $\Delta t_{\mathbf{j}}^{n_{\mathbf{j}}}$ of point $\mathbf{j}$ to be determined by an average of its neighbors time-levels. This averaging avoids the need of examining the stability condition before each time-step (as in the CA case) on one hand, and "smears" the time-levels of the grid points on the other hand. Therefore, the TS algorithm can be implemented efficiently, for instance, on a multiprocessor in which the processors are not necessarily identical, as long as their speeds do not differ too much, so that the "smearing" effect dominates the "gapping" effect. Note that speed differences can occur not only due to differences in the hardware configuration, but also due to unbalanced workload, which may be caused by the geometry or the specification of the problem.

In the next section we discuss in detail the consistency, stability, and convergence, as well as the *Time-Stabilizing* property of our new scheme.

3 Scheme Analysis

We proceed by investigating the numerical properties of our TS scheme, with variable Δt, given by Eq. (7).

Lemma 1 *The finite difference scheme of Eq. (7) is consistent with Eq. (1) if* $\forall \mathbf{j}, n_{\mathbf{j}}, l$ $\sum_{l=1}^{d} \frac{\delta_l^2 t}{(\Delta x_l)^2} \neq -1$ *and* $t_{\pm l} - t = O(\Delta x_l{}^2)$ *so that* $\frac{1}{K} = O(1)$ *as* $\Delta x_l \to 0\ \forall l$.

Proof: By Taylor expansion we obtain,

$$\begin{aligned}
\bar{u}_{-l} &= u_{-l} + (t_{-l} - t)(\tfrac{\partial u}{\partial t})_{-l} + O((t_{-l} - t)^2) \\
\bar{u}_{+l} &= u_{+l} + (t_{+l} - t)(\tfrac{\partial u}{\partial t})_{+l} + O((t_{+l} - t)^2) \\
(\tfrac{\partial u}{\partial t})_{-l} &= (\tfrac{\partial u}{\partial t}) - (\Delta x_l)(\tfrac{\partial^2 u}{\partial t \partial x_l}) + O((\Delta x_l)^2) \\
(\tfrac{\partial u}{\partial t})_{+l} &= (\tfrac{\partial u}{\partial t}) + (\Delta x_l)(\tfrac{\partial^2 u}{\partial t \partial x_l}) + O((\Delta x_l)^2)
\end{aligned}$$

where

$$(\frac{\partial u}{\partial t})_{\pm l} = (\frac{\partial u}{\partial t})_{\mathbf{j} \pm \mathbf{1}_l}(t)$$

Then, Eq. (7) becomes

$$\begin{aligned}
\frac{u^+ - u}{\Delta t_{\mathbf{j}}^{n_{\mathbf{j}}}} = \frac{1}{K} \sum_{l=0}^{d} \frac{a_l^2}{(\Delta x_l)^2} \quad & [\delta_l^2 u + (\delta_l^2 t)(\tfrac{\partial u}{\partial t}) + (\Delta x_l)(t_{+l} - t_{-l})(\tfrac{\partial^2 u}{\partial x_l \partial t}) \\
& + O(t_{-l} - t)^2 + O(t_{+l} - t)^2] + \tfrac{1}{K} F
\end{aligned}$$

Or

$$\begin{aligned}
\frac{u^+ - u}{\Delta t_{\mathbf{j}}^{n_{\mathbf{j}}}} &= \tfrac{1}{K}[\textstyle\sum_{l=0}^{d} a_l^2 \frac{\delta_l^2 u}{(\Delta x_l)^2} + (K-1)(\frac{\partial u}{\partial t})] + \frac{1}{K} \sum_{l=0}^{d} \frac{a_l^2}{(\Delta x_l)^2}[(\Delta x_l)(t_{+l} - t_{-l})(\frac{\partial^2 u}{\partial x_l \partial t}) + \\
& + O(t_{-l} - t)^2 + O(t_{+l} - t)^2] + \tfrac{1}{K} F
\end{aligned}$$

Therefore, under the above assumptions, Eq. (7) is consistent with Eq. (1) and the truncation error is $\sum_{l=1}^{d} O(\Delta x_l) + O(\Delta t_{\mathbf{j}}^{n_{\mathbf{j}}})$ ∎

We note that for the case of [2] and [3] in which Δt is constant, $t_{\pm l} = (n_{\pm l})\Delta t$ where $n_{\pm l}$ is the counter value. In this case, $t_{\pm l} - t = O(\Delta x_l{}^2)$ means that $\frac{\Delta t (n_{\pm l} - n)}{(\Delta x_l)^2} = O(1)$ and since $\frac{\Delta t}{(\Delta x_l)^2} = r_l = const$ in there, it is only required that $(n_{\pm l} - n)$ is uniformly bounded.

In addition, we emphasize that the requirement $t_{\pm l} - t = O(\Delta x_l)^2$ can be reduced to the requirement that $\frac{t_{\pm l} - t}{\Delta x_l} \to 0$ as $\Delta x_l \to 0$. This can be viewed as $\frac{\partial t(\vec{x})}{\partial x_l} \to 0$ when $\Delta x_l \to 0$ $(1 \leq l \leq d)$ where $\vec{x}$ is the position vector and $t(\vec{x})$ is a surface of the problem time for a given spatial mesh-size. As shall be discussed later, this property along with the boundness of $\frac{\partial t}{\partial x}$ as $n_{\mathbf{j}} \to \infty$ which we call the *Time-Stabilizing* property, has been verified by numerical simulations. It is also proved, under certain simplifications, in lemma 5 and is due to the following stability restriction on $\Delta t_{\mathbf{j}}^{n_{\mathbf{j}}}$:

Lemma 2 *A sufficient stability condition for Eq. (7) is,*

$$\Delta t_{\mathbf{j}}^{n_{\mathbf{j}}} = \frac{c}{2}\left(\frac{K}{\sum_{l=1}^{d} \frac{a_l^2}{(\Delta x_l)^2}}\right) \qquad 0 < c \leq 1 \tag{8}$$

Proof: Assume that some perturbations in the initial values of all grid points occurred. We wish to examine the effect of this perturbation on the updated values at the grid points. Let $e = \tilde{u} - u$ where $\tilde{u}$ and u are the perturbed and exact solutions of Eq. (7), respectively.

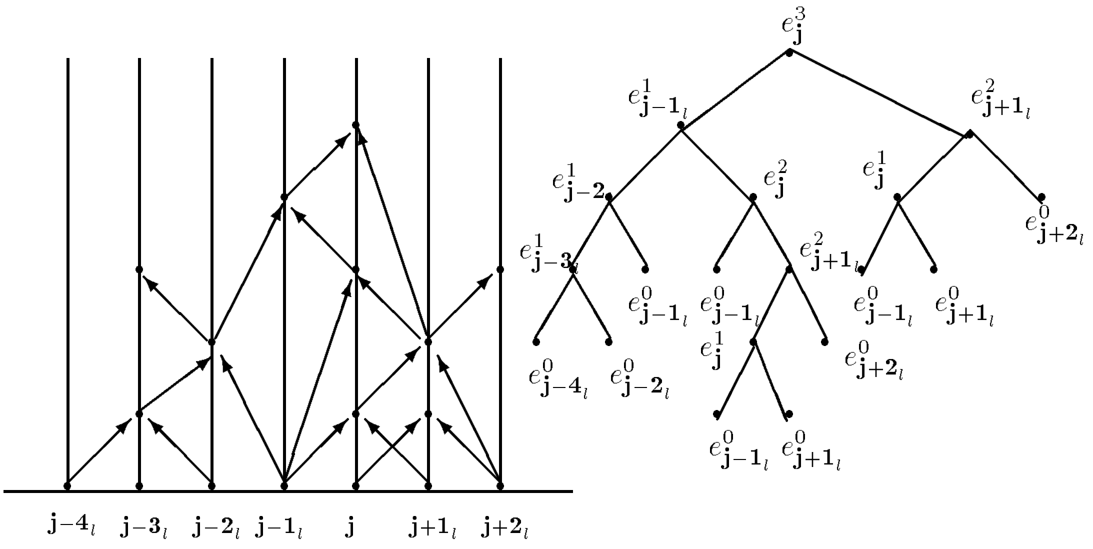
Then,

$$\begin{aligned} e^{+} &= e + \frac{\Delta t_{\mathbf{j}}^{n_{\mathbf{j}}}}{K} \sum_{l=1}^{d} \frac{a_l^2 \bar{\delta}_l^2 e}{(\Delta x_l)^2} = \\ &= [1 - \frac{2\Delta t_{\mathbf{j}}^{n_{\mathbf{j}}}}{K} \sum_{l=1}^{d} \frac{a_l^2}{(\Delta x_l)^2}]e + \frac{\Delta t_{\mathbf{j}}^{n_{\mathbf{j}}}}{K} \sum_{l=1}^{d} \frac{a_l^2}{(\Delta x_l)^2}[\bar{e}_{-l} + \bar{e}_{+l}] = \\ &= (1-c)e + \frac{c}{2\sum_{l=1}^{d} \frac{a_l^2}{(\Delta x_l)^2}} \sum_{l=1}^{d} \frac{a_l^2}{(\Delta x_l)^2}[\bar{e}_{-l} + \bar{e}_{+l}] \end{aligned}$$

Equation (8) assures that the coefficients of e, e_{-l} and e_{+l}, in the last equation, are all positive, and the sum of their absolute values is one. Therefore,

$$\begin{aligned} |e^{+}| &= |(1-c)e + \frac{c}{2\sum_{l=1}^{d} \frac{a_l^2}{(\Delta x_l)^2}} \sum_{l=1}^{d} \frac{a_l^2}{(\Delta x_l)^2}[\bar{e}_{-l} + \bar{e}_{+l}]| \leq \\ &\leq \max\{|e|, |\bar{e}_{-l}|, |\bar{e}_{+l}|\} \qquad 1 \leq l \leq d \end{aligned}$$

This equation is satisfied, whenever a value is updated. We may apply the last inequality recursively, following the "parents" of each grid point which maximizes $|e|$, until the initial values are reached. This may be viewed as going down a 3-tree along the reversed track of the updating track. When $c = 1$ we have a binary tree (see Fig. 2).



(a) (b)

Figure 2: (a) Updating (b) The associated binary tree ($c = 1$) for grid point $\mathbf{j}$ in direction l

Then we get

$$|e^{+}| \leq \max_k |e_k^0|$$

Hence, the scheme of Eq. (7) is stable. ∎

Note the following:

- From our consistency requirement $t_{\pm l} - t = O(\Delta {x_l}^2)$ we have that

$$\delta_l^2 t = (t_{+l} - t) + (t_{-l} - t) = O(\Delta {x_l}^2)$$

From Eqs. (8) and (5b) we then find that:

$$\Delta t_{\mathbf{j}}^{n_{\mathbf{j}}} = t_{\mathbf{j}}^{n_{\mathbf{j}}+1} - t_{\mathbf{j}}^{n_{\mathbf{j}}} = \frac{c}{2\sum_{l=1}^{d} \frac{a_l^2}{(\Delta x_l)^2}} + \frac{c}{2\sum_{l=1}^{d} \frac{a_l^2}{(\Delta x_l)^2}} \sum_{l=1}^{d} \frac{a_l^2 \delta_l^2 t_{\mathbf{j}}^{n_{\mathbf{j}}}}{(\Delta x_l)^2} = O\left(\frac{1}{\sum_{l=1}^{d} \frac{a_l^2}{(\Delta x_l)^2}}\right)$$

Hence, $\Delta t_{\mathbf{j}}^{n_{\mathbf{j}}} \to 0$, as $\Delta x_l \to 0$. Furthermore, $\frac{\Delta t_{\mathbf{j}}^{n_{\mathbf{j}}}}{\Delta x_l} \to 0$ as $\Delta x_l \to 0 \ \forall l$. Therefore, if the above requirements are fulfilled, then for consistency and convergence of our scheme we need to consider the relevant limits only as $\Delta x_l \to 0$ $(1 \leq l \leq d)$ and disregard the additional requirement that $\Delta t_{\mathbf{j}}^{n_{\mathbf{j}}} \to 0$.

- The choice of $\Delta t_{\mathbf{j}}^{n_{\mathbf{j}}}$ according to Eq. (8) also eliminates the possibility of zero division in Eq. (7).

- $\Delta t_{\mathbf{j}}^{n_{\mathbf{j}}}$ in Eq. (8) may become zero or even negative which means that the status of the neighboring points prevents further advance in time.

By the choice of $\Delta t_{\mathbf{j}}^{n_{\mathbf{j}}}$ according to Eq. (8), our TS scheme becomes,

$$u^+ = (1-c)u + \frac{c}{\sum_{l=1}^{d} \frac{a_l^2}{(\Delta x_l)^2}} \sum_{l=1}^{d} \frac{a_l^2 \bar{\mu}_l u}{(\Delta x_l)^2} + \frac{c}{2\sum_{l=1}^{d} \frac{a_l^2}{(\Delta x_l)^2}} F \tag{9a}$$

$$t^+ = (1-c)t + \frac{c}{2\sum_{l=1}^{d} \frac{a_l^2}{(\Delta x_l)^2}} + \frac{c}{\sum_{l=1}^{d} \frac{a_l^2}{(\Delta x_l)^2}} \sum_{l=1}^{d} \frac{a_l^2 \bar{\mu}_l t}{(\Delta x_l)^2} \qquad 0 < c \leq 1 \tag{9b}$$

when

$$\bar{\mu}_l v = \frac{\bar{v}_{+l} + \bar{v}_{-l}}{2} \tag{9c}$$

In one dimension Eqs. (9a) and (9b) become,

$$u^+ = (1-c)u + c\frac{\bar{u}_+ + \bar{u}_-}{2} + \frac{c}{2}\frac{(\Delta x)^2}{a^2} F \tag{10a}$$

$$t^+ = (1-c)t + \frac{c(\Delta x)^2}{2a^2} + c\frac{t_+ + t_-}{2} \tag{10b}$$

As will be shown later, the optimal parameter value is $c = 1$, so that u^+ and t^+ are determined by the weighted averages of the corresponding neighbors values along each spatial axis.

Hence, Eq. (7) which is a scheme for u with variable coefficients, is replaced by two schemes (Eqs. (9a) and (9b)) with constant coefficients; one scheme for u and the other for t.

Lemma 3 *Assume:*

$$t_{\pm l} - t = O({\Delta x_l}^2) \qquad \forall \mathbf{j}, n_{\mathbf{j}}, l \quad , \tag{11a}$$

$$\sum_{l=1}^{d} \frac{a_l^2 \delta_l^2 t_{\mathbf{j}}^{n_{\mathbf{j}}}}{(\Delta x_l)^2} \neq -1 \quad , \tag{11b}$$

$$\frac{1}{K} = O(1) \quad , \tag{11c}$$

and

$$\Delta t_{\mathbf{j}}^{n_{\mathbf{j}}} = \frac{c}{2}\left(\frac{K}{\sum_{l=1}^{d} \frac{a_l^2}{(\Delta x_l)^2}}\right) \qquad 0 < c \leq 1. \tag{11d}$$

Then the scheme of Eq. (7) converges as $\Delta x_l \to 0 \quad \forall l$.

Proof: First note that by applying Eq. (9b) repeatedly until the time-level $t = 0$, we find that

$$t_{\mathbf{j}}^{n_{\mathbf{j}}} = B_{\mathbf{j}}^{n_{\mathbf{j}}} \frac{c}{2\sum_{l=1}^{d} \frac{a_l^2}{(\Delta x_l)^2}} \tag{12}$$

where, $B_{\mathbf{j}}^{n_{\mathbf{j}}}$ depends upon the ratios $\frac{\Delta x_k}{\Delta x_l}$ $(1 \leq k, l \leq d)$ and upon the reversed tracks along the tree corresponding to the grid point $\mathbf{j}$ from $t = t_{\mathbf{j}}^{n_{\mathbf{j}}}$ level to $t = 0$ level. However, $B_{\mathbf{j}}^{n_{\mathbf{j}}}$ is independent of Δx_l. We consider the convergence at the grid point $\mathbf{j}$ for $t = t_{\mathbf{j}}^{n_{\mathbf{j}}}$ as $\Delta x_l \to 0$ $\forall l$ when $t_{\mathbf{j}}^{n_{\mathbf{j}}}$ of Eq. (12) is held fixed. Recall that Eq. (11a) and (11d) ensure that $\Delta t_{\mathbf{j}}^{n_{\mathbf{j}}} \to 0$, if $\Delta x_l \to 0$ $(1 \leq l \leq d)$. Therefore, convergence is considered when $\Delta x_l \to 0$ $\forall l$, and under the above restrictions, the requirement that $\Delta t_{\mathbf{j}}^{n_{\mathbf{j}}} \to 0$, is automatically fulfilled.

This convergence can be achieved, for example, in the following way. For a fixed grid point $\mathbf{j}$, and for some initial specified Δx_l $(1 \leq l \leq d)$ let $T_{\mathbf{j}}^k$ be an increasing monotonic sequence of problem-times at which the solution is evaluated by some multiprocessor machine, such that $T_{\mathbf{j}}^k \to \infty$ as $k \to \infty$ and $T_{\mathbf{j}}^k > t_{\mathbf{j}}^{n_{\mathbf{j}}}$ $\forall k$ where $t_{\mathbf{j}}^{n_{\mathbf{j}}}$ is the time value that has to be held constant when $\Delta x_l \to 0$. It follows from Eq. (12), when $t_{\mathbf{j}}^{n_{\mathbf{j}}}$ is replaced by $T_{\mathbf{j}}^k$, that $T_{\mathbf{j}}^k$ are increasing monotonic functions of Δx_l. Therefore, for each k, Δx_l can be reduced to Δx_l^k by a corresponding factor as to ensure that $\mathbf{j}$ is always a grid point and that $T_{\mathbf{j}}^k = t_{\mathbf{j}}^{n_{\mathbf{j}}}$ $\forall k$. We thereby obtain a decreasing sequence $\Delta x_l^k \to 0$ as $k \to \infty$ $(1 \leq l \leq d)$, and a corresponding increasing sequence $B_{\mathbf{j}_k}^{n_{\mathbf{j}}} \to \infty$ as $k \to \infty$, such that when substituted in Eq. (12), for any k, they all yield the same time-level $t = t_{\mathbf{j}}^{n_{\mathbf{j}}}$.

With this process, the first and last grid points along each spatial axis may not fall exactly on the original boundaries. In this case the boundary may be shifted to the positive or negative spatial direction to match exactly the first and last grid points along the axis. The corresponding error is, of course, $O(\Delta x_l)$ $(1 \leq l \leq d)$ which, as shall be shown below, is the order of the overall error.

To prove convergence, let $\varepsilon = u - v$ where u is the solution of Eq. (9a) and v is the exact solution of Eq. (1) with the corresponding initial/boundary conditions. By Taylor series with remainder (see the proof of lemma 1), using Eqs. (8) and (11c) we have under the above assumptions:

$$\overset{n}{v}_{\mathbf{j}}^{\mathbf{j}+1} = (1-c)\overset{n}{v}_{\mathbf{j}}^{\mathbf{j}} + \frac{c}{\sum_l \frac{a_l^2}{(\Delta x_l)^2}} \sum_l \frac{a_l^2 \bar{\mu}_l v}{(\Delta x_l)^2} + \frac{c}{2\sum_l \frac{a_l^2}{(\Delta x_l)^2}} F + K_1 \frac{c}{2} \frac{K}{\sum_l \frac{a_l^2}{(\Delta x_l)^2}} \tag{13a}$$

in which

$$K_1 = \sum_l [O(\Delta x_l)] + O[\frac{1}{\sum \frac{a_l^2}{(\Delta x_l)^2}}] \tag{13b}$$

By subtracting Eq. (13a) from Eq. (9a) and applying the triangular inequality, we obtain

$$|\overset{n}{\varepsilon}_{\mathbf{j}}^{\mathbf{j}+1}| \leq (1-c)|\overset{n}{\varepsilon}_{\mathbf{j}}^{\mathbf{j}}| + \frac{c}{\sum_l \frac{a_l^2}{(\Delta x_l)^2}} \sum_l \frac{a_l^2 \bar{\mu}_l |\varepsilon|}{(\Delta x_l)^2} + \frac{c}{2} \frac{M}{\sum_l \frac{a_l^2}{(\Delta x_l)^2}} \sum_{s=1}^{d} (\Delta x_s)$$

where M is an upper bound on the magnitudes of the solution second order derivates.

As for $\overset{n}{t}_{\mathbf{j}}^{\mathbf{j}}$, since $\varepsilon_{\mathbf{j}}^0 = 0\ \forall \mathbf{j}$, we now get by applying the last inequality recursively,

$$|\overset{n}{\varepsilon}_{\mathbf{j}}^{\mathbf{j}}| \leq \overset{n}{B}_{\mathbf{j}}^{\mathbf{j}} \frac{c}{2} \frac{M}{\sum_l \frac{a_l^2}{(\Delta x_l)^2}} \sum_s (\Delta x_s)$$

where $\overset{n}{B}_{\mathbf{j}}^{\mathbf{j}}$ is the *same* as in Eq. (12), so that

$$|\overset{n}{\varepsilon}_{\mathbf{j}}^{\mathbf{j}}| \leq \overset{n}{t}_{\mathbf{j}}^{\mathbf{j}} M \sum_s (\Delta x_s)$$

Therefore, at the grid point $\mathbf{j}$ and for fixed $\overset{n}{t}_{\mathbf{j}}^{\mathbf{j}}$, $\overset{n}{\varepsilon}_{\mathbf{j}}^{\mathbf{j}} \rightarrow 0$ as $\Delta x_l \rightarrow 0$ $(1 \leq l \leq d)$ and the convergence is proved. ∎

It can be shown easily that the synchronous version of the time-difference scheme given by Eq. (9b) is stable. This is illustrated by Fig. 3 which plots the error in time-level $t_{\mathbf{j}}^n$ versus x in the one-dimensional case for different number of time-steps and for initial sinusoidal disturbance when Eq. (9b) is used synchronously with $c = 1$. As noticed, the error approaches zero as $n \rightarrow \infty$. Moreover, the asynchronous version of Eq. (9b) is stable as well:

Lemma 4 *The time difference equation Eq. (9b) is stable.*

Proof: Analogous to the proof of lemma 2. ∎

We shall now examine carefully the *Time-Stabilizing* property of our scheme: $|\frac{\partial t(\vec{x})}{\partial x_l}|$ is bounded as the time-step number increases and $\frac{\partial t(\vec{x})}{\partial x_l} \rightarrow 0$ as $\Delta x_l \rightarrow 0$, where $\vec{x}$ is the position vector and $t(\vec{x})$ is a surface of the problem time for a fixed spatial mesh-size Δx_l $(1 \leq l \leq d)$. This is illustrated clearly by Figs. 4, 5, and 6 which plot the relative time-level $t - t_s$ versus x-coordinate in the one-dimensional case where t_s is the time-level of the "slowest" grid point, and for $\Delta x = 0.05263$, 0.03448, and 0.02564, respectively. In all the three cases, each processor is assigned a single grid point. In these simulations, we scan all grid points

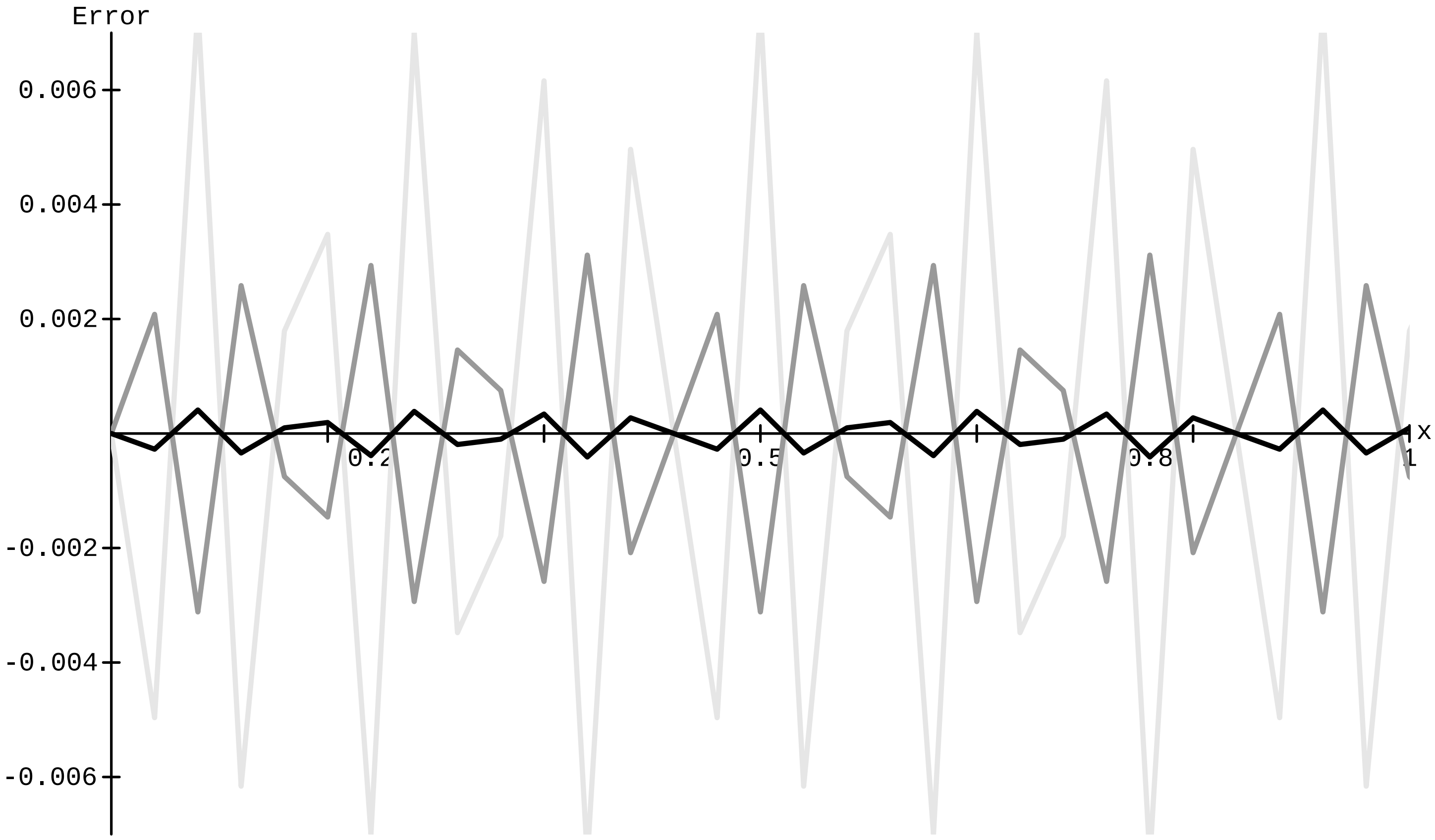


Figure 3: Decay of initial sinusoidal perturbation in the synchronous version of Eq. (9b) for the 1-D case. Each processor is assigned one grid point and $\Delta x = 0.0204$. The three snapshots are taken at the second, at the sixth, and at the twelveth time-steps.

iteratively, and determine which can be updated. All grid points are updated simultaneously at the same rate (every 9^{th} iteration), except the middle point ($x = 0.5$), which is updated at 10% "slower" rate (every 10^{th} iteration). This iterative scanning is considered as ticks of some outside global clock that determine the behavior of the system. We observe that as Δx decreases, the time-level flattens and the time-level differences between the grid points diminish. Furthermore, for a fixed mesh-size, a "steady-state" is reached in the sense that $|t_{\mathbf{i}} - t_{\mathbf{j}}|$ for $\mathbf{i} \neq \mathbf{j}$ is bounded as $n_{\mathbf{j}} \to \infty$. The same phenomena occur when each processor is assigned a group of grid points as illustrated by Fig. 7 and Fig. 8. In the last two cases, a group of grid points is assigned to each processor, and t_s is the time level of the "slowest" processor. All processors operate at the same rate, except the 11^{th} one which is 10% "slower". We emphasize, however, that for relatively large speed-or load-differences, the *Time-Stabilizing* property will disappear.

In the case of one grid point per processor, when the "slow" grid point is updated, its time-level becomes higher than that of all the others (since it is the average of neighboring time-levels incremented by $\frac{1}{2}(\Delta x)^2$) and a global maximum is attained at the "slow" grid point (see Figs. 4-6). Note, however, that if the snapshot is taken immediately after the fast processor updating, then a local minimum is obtained at the "slow" grid point and global maxima at its left and right neighbors.

In the case of several grid points per processor, there is an interaction between two opposing mechanisms. The inner grid points of the "slow" processor tend to lower the problem time-level while its edge points (which are the most affected by the fast grid points of adjacent processors) tend to raise it. If there are a relatively small number of grid points per processor, the effect of the edge points is dominant. Then the time-level attains a global maxima at the edge points of the "slow" processor or at the closest edge points of the fast processors (depending on whether the snapshot is taken immediately after an updating of the "slow" processor or immediately after an updating of the fast processors). In addition, a local minimum is attained at the midpoint of the "slow" processor (see Fig. 7). If there are a relatively large number of grid points per processor, the effect of the inner "slow" grid points becomes dominant and the problem time-level attains a funnel shape as illustrated by Fig. 8.

As indicated in [3], the CA algorithm can be applied only when the differences between time-levels of adjacent grid points are bounded. This can never happen, for example, when implemented on a parallel machine with processors of different speeds or different workload as in the example of Fig. 4 through Fig. 8. However, the *Time-Stabilizing* property of our new scheme enables it to be implemented in such circumstances, provided that the speed/workload differences are not too large to destroy this stabilizing effect.

We now wish to examine the time difference scheme (9b) from a *global* point of view. At a real time τ_0 all grid points have initial values for all the variables $u_{\mathbf{j}}$ in the corresponding problem time $t_{\mathbf{j}}$. After a real time passage of $\tau_1 - \tau_0$ units, the "fastest" grid point has completed N_f steps while the "slowest" grid point has completed only $N_s < N_f$ steps.

If we take a shorter time interval such that the fastest grid point has completed one but not more than one updating, then we can write the time equation in the following global

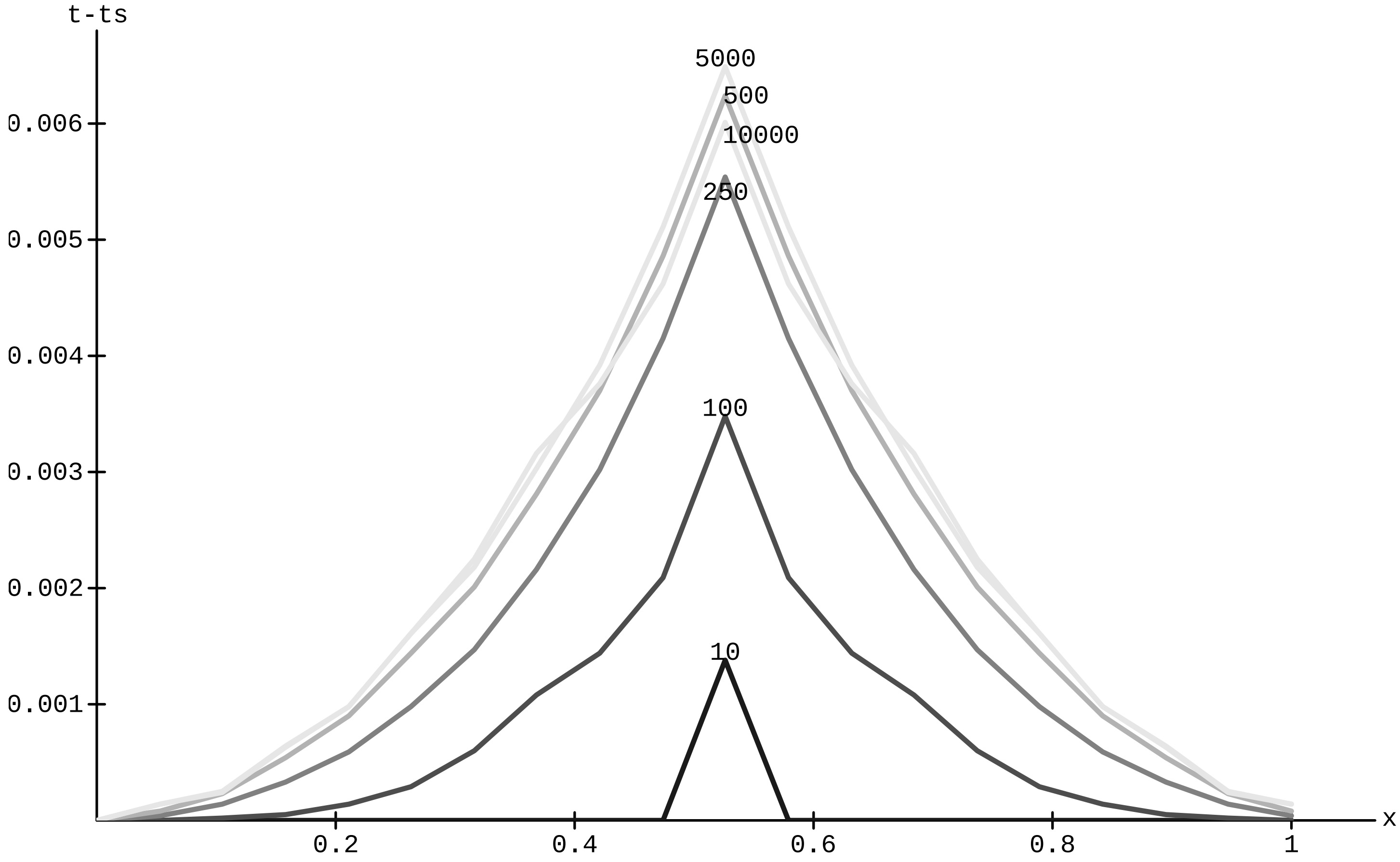


Figure 4: Dependence of t relative to the time-level t_s of the "slowest" grid point on the spatial coordinate for the 1-D case. Each processor is assigned one grid point, the middle grid point ($x = 0.5$) is updated at 10% "slower" rate than the others, and $\Delta x = 0.05263$. The numbers indicate ordinal numbers displayed by a global counter of external equal time-intervals.

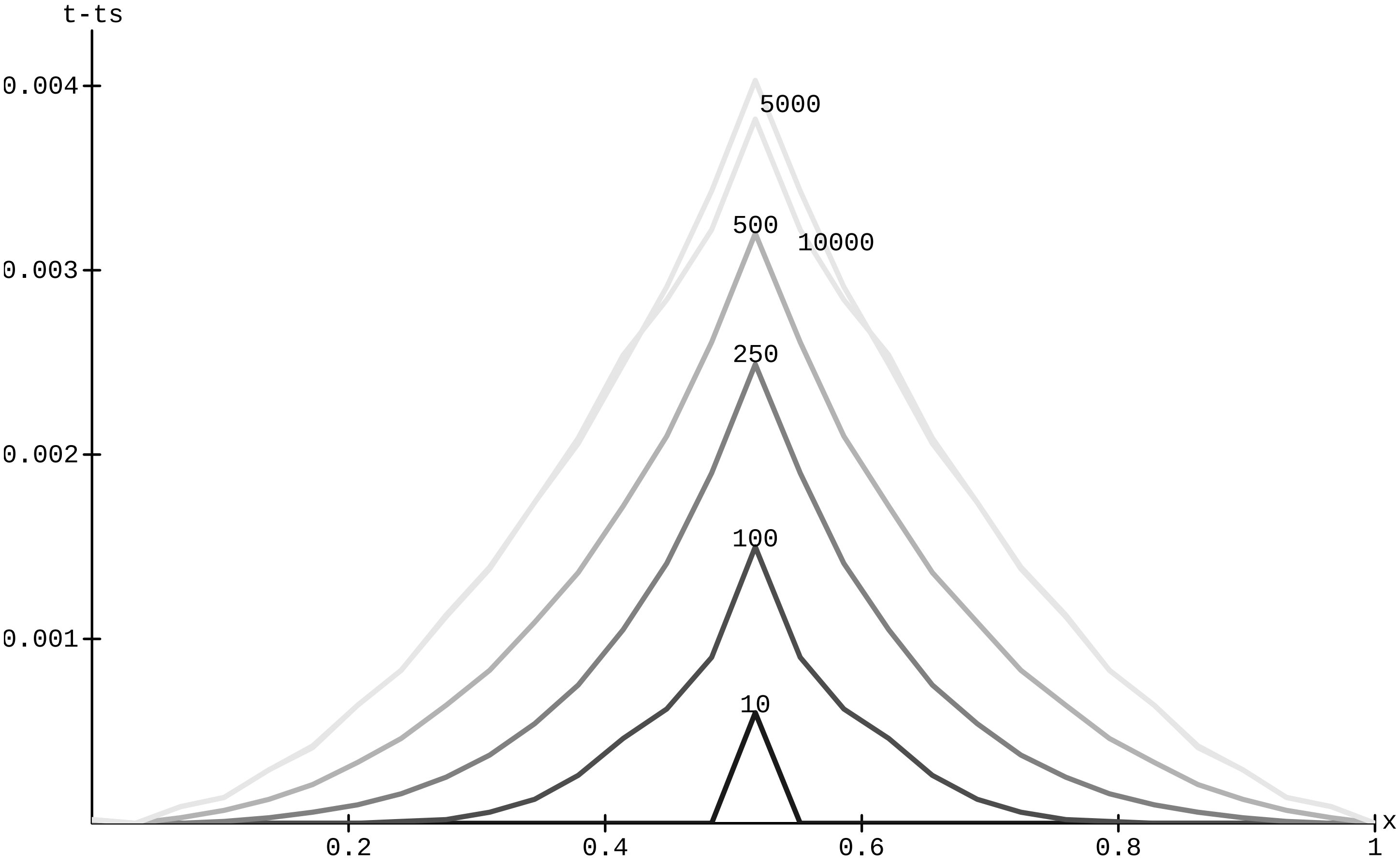


Figure 5: Dependence of t relative to the time-level t_s of the "slowest" grid point on the spatial coordinate for the 1-D case, when $\Delta x = 0.03448$. Other parameters are as in Fig.4.

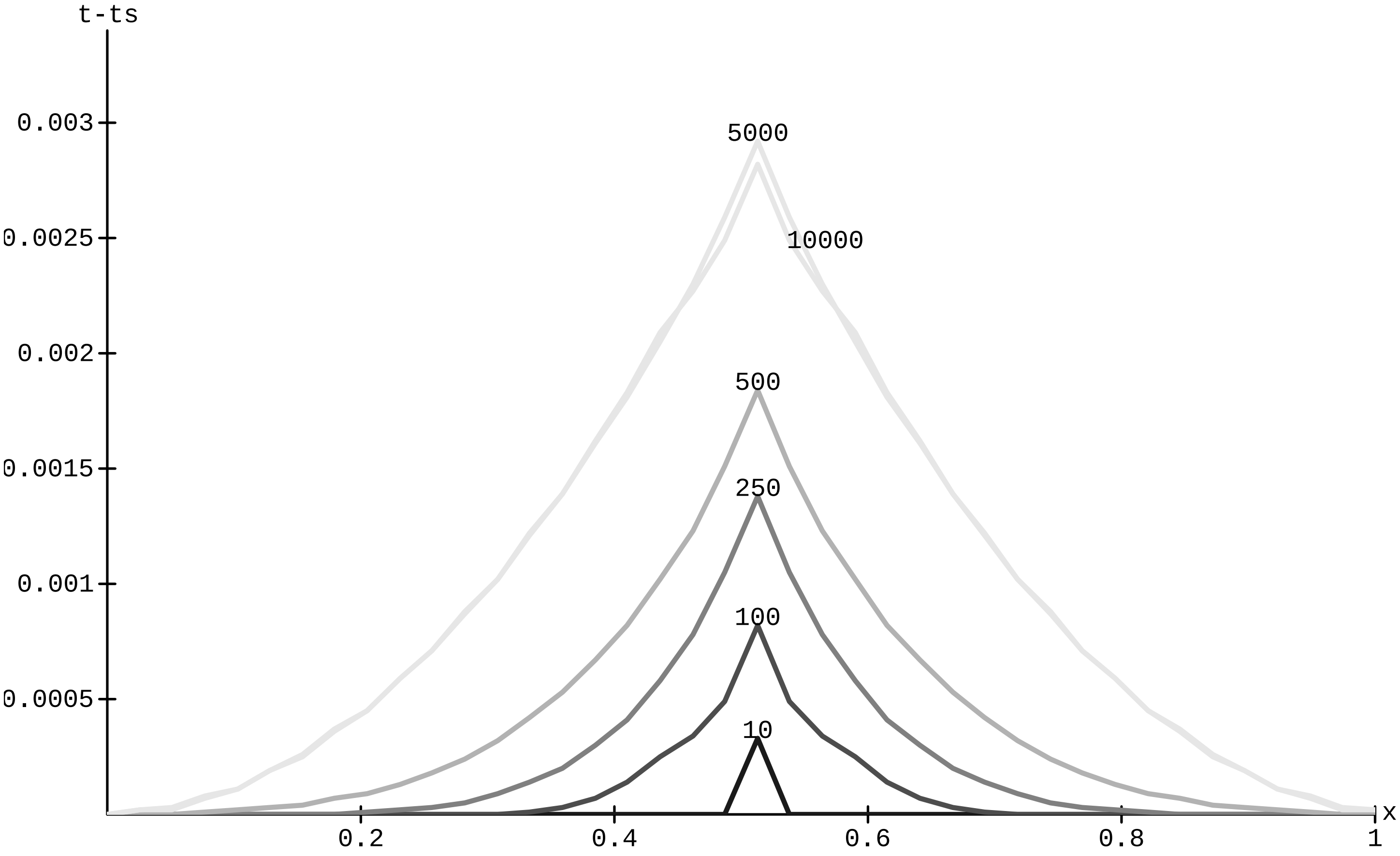


Figure 6: Dependence of t relative to the time-level t_s of the "slowest" grid point on the spatial coordinate for the 1-D case, when $\Delta x = 0.02564$. Other parameters are as in Fig.4.

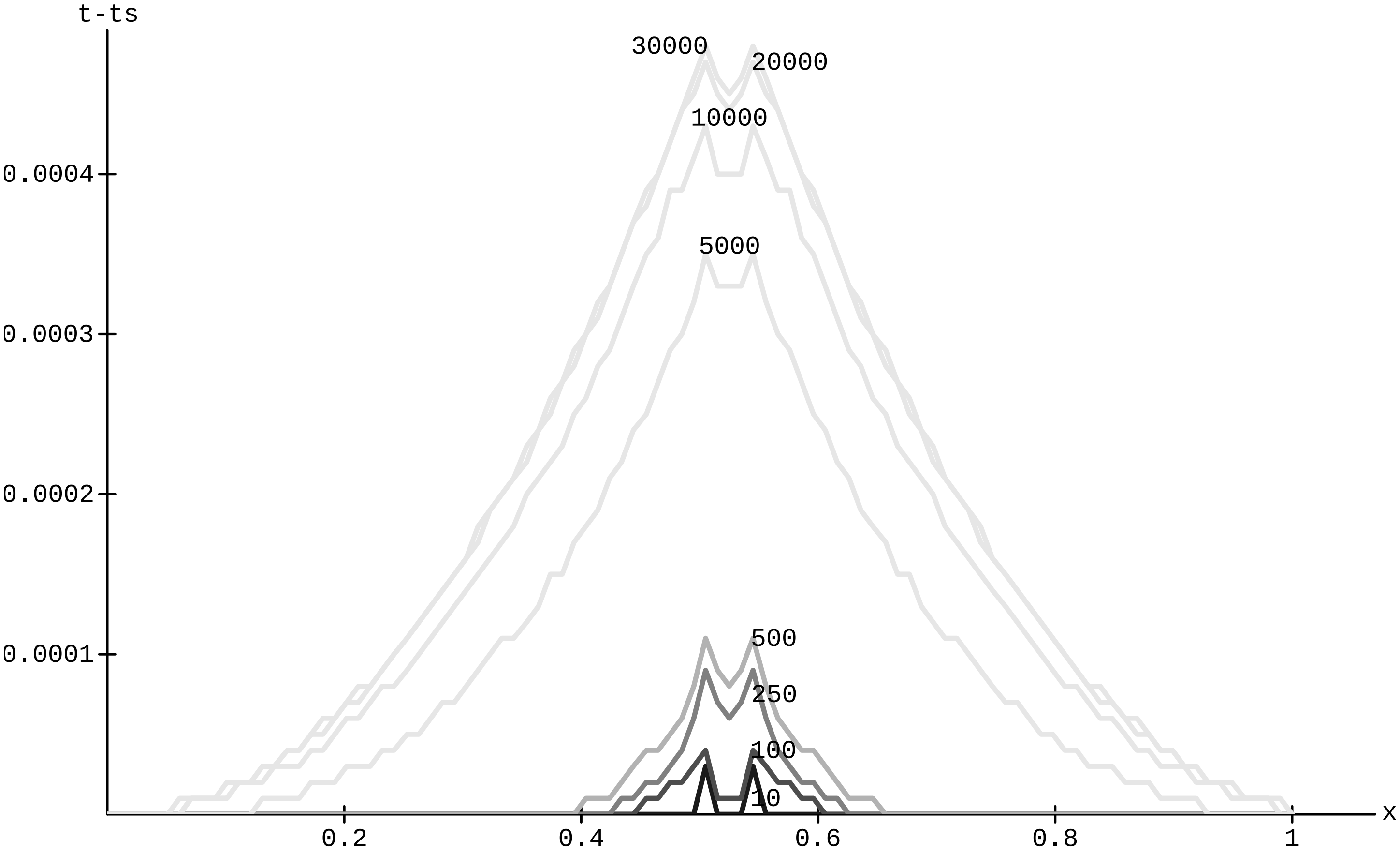


Figure 7: Simulation of 1-D problem by 20 processors, 5 points per processor, where the 11^{th} processor is 10% "slower" than the rest, and $\Delta x = 0.01010$. Other parameters are as in Fig. 4.

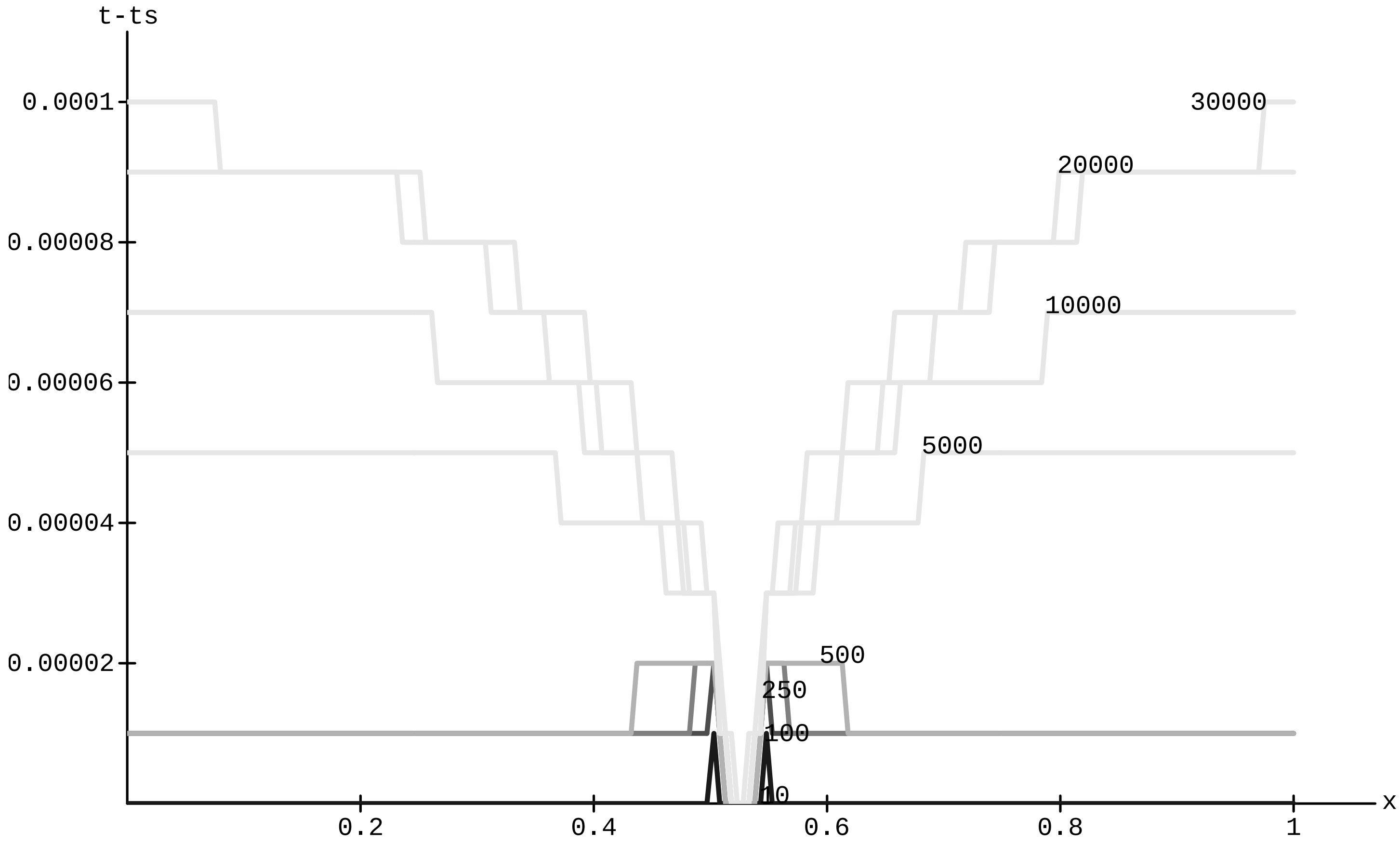


Figure 8: Simulation of 1-D prob. by 20 processors, 10 points per processor, where the 11^{th} processor is 10% "slower" than the rest, and $\Delta x = 0.00502$. Other parameters are as in Fig. 4.

form:

$$t^{+}=t+\frac{c}{2\sum_{l=1}^{d}\frac{a_l^2}{(\Delta x_l)^2}}\sum_{l=1}^{d}\frac{a_l^2\delta_l^2 t}{(\Delta x_l)^2}+\phi-\mathcal{F}_{\mathbf{j}} \qquad (14a)$$

where

$$\phi=\frac{c}{2\sum_{l=1}^{d}(\frac{a_l^2}{\Delta x_l})^2} \qquad (14b)$$

and

$$\mathcal{F}_{\mathbf{j}}=\begin{cases} 0 & \text{if updating occurred} \\ \phi+\frac{c}{2\sum_{l=1}^{d}\frac{a_l^2}{(\Delta x_l)^2}}\sum_{l=1}^{d}\frac{a_l^2\delta_l^2 t}{(\Delta x_l)^2} & \text{otherwise} \end{cases} \qquad (14c)$$

Hence, we may consider a sequence of clock-times, and a corresponding global sequence of problem-times, such that between any two consequent clock-times, the fastest grid point (which may be different at different clock-time intervals) has completed one and only one updating. However, this situation makes the global analysis extremely difficult, since we have to deal with a non-constant coefficient equation. Furthermore, this equation is essentially fed by a non-linear random input, as $\mathcal{F}_{\mathbf{j}}$ depends on the solution $t_{\mathbf{j}}$. Note that each time the equation is used with $\mathcal{F}_{\mathbf{j}}=0$, the central difference operator in Eq. (14a) will tend to smooth the function it operates on. Suppose now that smoothing is being performed always for all the grid points. If appropriate boundary conditions are specified at boundary grid points (i.e. homogeneous Neumann boundary conditions), any perturbation in the initial problem time profile will die out eventually and we will obtain a smooth front. However, we have to account for the fact that not all the grid points are assigned smoothing all the time.

In order to simplify our analysis we shall make the following simplifying assumptions:

1. Each grid point is updated continuously.
2. To account for a distinct number of smoothing steps, we will make the severe assumption that smoothing is performed for all the grid points, at the rate of the slowest one.
3. Grid point $\mathbf{j}$ will have N_s equal increments $\phi\frac{N_{\mathbf{j}}}{N_s}$.

Lemma 5 *With the above assumptions, Eq. (9b) is Time-Stabilizing, meaning that with initial zero time-level,* $\frac{\partial t}{\partial x_l}\to 0$ $(1\le l\le d)$ *as* $\Delta x_l\to 0$, *and* $|\frac{\partial t}{\partial x_l}|$ *is bounded as* $n\to\infty$.

Proof: Under the above assumptions, the time-difference equation (9b) is simplified into the following:

$$t_{\mathbf{j}}^{n+1}=t_{\mathbf{j}}^{n}+\frac{c}{2\sum_{l=1}^{d}\frac{a_l^2}{(\Delta x_l)^2}}\left(\sum_{l=1}^{d}\frac{a_l^2(\delta_l^2 t)_{\mathbf{j}}^n}{(\Delta x_l)^2}+g_{\mathbf{j}}^n\right) \quad 0\le c\le 1 \qquad (15)$$

in which $g_{\mathbf{j}}^n=\frac{N_{\mathbf{j}}}{N_s}$ is the ratio of the corresponding computation speeds (and it is independent of Δx_l), and n is a global time-step number value. We proceed to analyze the behavior of this equation as $\Delta x_l\to 0$ and as the number of time steps approaches infinity.

At each time step we perform a Discrete Cosine Fourier Transform with the Neumann boundary condition:

$$t_{\mathbf{j}}^{n} = \mathrm{Re}\{\sum_{k_1,\ldots,k_d} \hat{t}_{\mathbf{k}}^{n} e^{i\mathbf{k}\cdot\vec{x}_{\mathbf{j}}}\}$$

where $\vec{x}_{\mathbf{j}}$ is the coordinates vector of grid point $\mathbf{j}$, $\mathbf{k} = (k_1, \ldots, k_d)$, $0 \leq k_l \leq J_l + 1$ and J_l $(1 \leq l \leq d)$ is the number of interior grid points along the l-axis. Then

$$\begin{aligned}\hat{t}_{\mathbf{k}}^{n+1} &= \hat{t}_{\mathbf{k}}^{n} + \frac{c}{2\sum_{l=1}^{d} \frac{a_l^2}{(\Delta x_l)^2}} \hat{g}_{\mathbf{k}}^{n} + \frac{c}{2\sum_{l=1}^{d} \frac{a_l^2}{(\Delta x_l)^2}} \sum_{l=1}^{d} a_l^2 \frac{-4\sin^2 \frac{k_l \Delta x_l}{2}}{(\Delta x_l)^2} \hat{t}_{\mathbf{k}}^{n} = \\ &= q_{\mathbf{k}} \hat{t}_{\mathbf{k}}^{n} + \frac{c}{2\sum_{l=1}^{d} \frac{a_l^2}{(\Delta x_l)^2}} \hat{g}_{\mathbf{k}}^{n}\end{aligned}$$

where,

$$q_{\mathbf{k}} = 1 - \frac{2c}{\sum_{l=1}^{d} \frac{a_l^2}{(\Delta x_s)^2}} \sum_{l=1}^{d} \frac{a_l^2 \sin^2(\frac{k_l \Delta x_l}{2})}{(\Delta x_l)^2}$$

is independent of t. Thus,

$$|\hat{t}_{\mathbf{k}}^{n+1}| \leq |q_{\mathbf{k}}||\hat{t}_{\mathbf{k}}^{n}| + |\frac{c}{2\sum_{l=1}^{d} \frac{a_l^2}{(\Delta x_l)^2}}||\hat{g}_{\mathbf{k}}^{n}|$$

Let $\tilde{g}_{\mathbf{k}}$ be the maximum in time of $|\hat{g}_{\mathbf{k}}^{n}|$ for each $\mathbf{k}$. It can be shown easily that $q_{\mathbf{k}} = 1$ if $\cos(k_l \Delta x_l) = 1 \quad \forall l$ or if $c = 0$, while $q_{\mathbf{k}} = -1$ if $\cos(k_l \Delta x_l) = -1 \quad \forall l$ and $c = 1$. Otherwise, $|q_{\mathbf{k}}| < 1$. Then by using the last inequality recursively we obtain

$$|\hat{t}_{\mathbf{k}}^{n}| \leq |\hat{t}_{\mathbf{k}}^{0}||q_{\mathbf{k}}|^n + \tilde{g}_{\mathbf{k}}(\frac{|q_{\mathbf{k}}|^n - 1}{|q_{\mathbf{k}}| - 1})(\frac{c}{2\sum_{s=1}^{d} \frac{a_l^2}{(\Delta x_s)^2}})$$

The first term on the right hand side vanishes in our case, since the initial time-level is zero so that $\frac{\partial t}{\partial x_l} \to 0$ as $\Delta x_l \to 0$ $(1 \leq l \leq d)$. Also if $|q_{\mathbf{k}}| < 1$ then for fixed spatial mesh sizes, $|\hat{t}_{\mathbf{k}}^{n}|$ is bounded as $n \to \infty$. ∎

4 Other Schemes

Our derivation of Eqs. (9a) and (9b) was based on Euler forward scheme (Eq. (2)). Similar derivations applied on other implicit or explicit scheme in which the value of the $(n_{\mathbf{j}} + 1)^{th}$ iteration at the grid point $\mathbf{j}$ is determined by its value at the previous (i.e ${n_{\mathbf{j}}}^{th}$) time-step and by the most recent value at the points $\mathbf{j} \pm \mathbf{1}_l$ $(1 \leq l \leq d)$ would lead to the same scheme namely Eqs. (9a) and (9b). In the case when it is determined by two or more levels' values of its neighbors (e.g Crank-Nicholson method), the corresponding consistent asynchronous scheme becomes much more complicated [1]. Thus, for example, Laasonen, and Saul'ev [10] schemes would lead to (9a) and (9b). However, DuFort-Frankel scheme would lead to a slightly different approximation.

In this case the multidimensional synchronous scheme is:

$$u_{\mathbf{j}}^{n+1} = u_{\mathbf{j}}^{n-1} + 2\sum_{l=1}^{d} r_l(u_{\mathbf{j}-1_l}^{n} - u_{\mathbf{j}}^{n-1} - u_{\mathbf{j}}^{n+1} + u_{\mathbf{j}+1_l}^{n}) + 2\Delta t_j^n F \qquad r_l = \frac{a_l^2\Delta t}{(\Delta x_l)^2}$$

The asynchronous scheme becomes:

$$\begin{aligned} u_{\mathbf{j}}(t_{\mathbf{j}}^{n_{\mathbf{j}}+1}) &= u_{\mathbf{j}}(t_{\mathbf{j}}^{n_{\mathbf{j}}-1}) + \\ &+ (t_{\mathbf{j}}^{n_{\mathbf{j}}} - t_{\mathbf{j}}^{n_{\mathbf{j}}-1})\textstyle\sum_{l=1}^{d} \frac{a_l^2}{(\Delta x_l)^2}[u_{\mathbf{j}-1_l}(t_{\mathbf{j}-1_l}^{n_{\mathbf{j}}}) - u_{\mathbf{j}}(t_{\mathbf{j}}^{n_{\mathbf{j}}-1}) - u_{\mathbf{j}}(t_{\mathbf{j}}^{n_{\mathbf{j}}+1}) + u_{\mathbf{j}+1_l}(t_{\mathbf{j}+1_l}^{n_{\mathbf{j}}})] + \\ &+ (t_{\mathbf{j}}^{n_{\mathbf{j}}+1} - t_{\mathbf{j}}^{n_{\mathbf{j}}-1})F \end{aligned}$$

and the corresponding time-stabilizing scheme is:

$$u^+ = u^- + \frac{t^+ - t^-}{K}\sum_{l=1}^{d}\frac{a_l^2}{(\Delta x_l)^2}(\bar{u}_{-l} - u^- - u^+ + \bar{u}_{+l}) + \frac{t^+ - t^-}{K}F$$

where

$$K = 1 + \sum_{l=1}^{d}\frac{a_l^2\delta_l^2 t_{\mathbf{j}}^{n_{\mathbf{j}}}}{(\Delta x_l)^2} - \sum_{l=1}^{d}(\frac{a_l}{\Delta x_l})^2(t^- - 2t + t^+),$$

$t^- = t_{\mathbf{j}}^{n_{\mathbf{j}}-1}$, and $u^- = u_{\mathbf{j}}(t_{\mathbf{j}}^{n_{\mathbf{j}}-1})$.

In its explicit form we get:

$$\begin{aligned} u^+ &= [\frac{K-(t^+-t^-)\sum\frac{a_l^2}{(\Delta x_l)^2}}{K+(t^+-t^-)\sum\frac{a_l^2}{(\Delta x_l)^2}}]u^- + \frac{t^+-t^-}{K+(t^+-t^-)\sum\frac{a_l^2}{(\Delta x_l)^2}}\textstyle\sum_{l=1}^{d}\frac{a_l^2}{(\Delta x_l)^2}(\bar{u}_{-l} + \bar{u}_{+l}) + \\ &+ \frac{t^+-t^-}{K+(t^+-t^-)\sum\frac{a_l^2}{(\Delta x_l)^2}}F \end{aligned}$$

Similarly to our analysis in the previous section, it can be shown that this scheme is convergent if

$$t_{\pm l} - t = O(\Delta x_l{}^2) \qquad \forall j, n_j, l \tag{16a}$$

$$\sum_{s=1}^{d}(\frac{a_s}{\Delta x_s})^2[\delta_s^2 t - (t^- - 2t + t^+)] \neq -1 \tag{16b}$$

and

$$t^+ - t- = \frac{c}{(2-c)\sum_{s=1}^{d}\frac{a_s^2}{(\Delta x_s)^2}}\{1 + \sum_{l=1}^{d}\frac{a_l^2[\delta_l^2 t - (t^- - 2t + t^+)]}{(\Delta x_l)^2}\} \qquad 0 < c \leq 1 \tag{16c}$$

In addition, the time finite difference equation is stable, and the scheme is time-stabilizing. Specifically, we obtain the following solution and time approximations:

$$u^+ = (1-c)u^- + \frac{c}{\sum_{l=1}^{d}\frac{a_l^2}{(\Delta x_l)^2}}\sum_{l=1}^{d}\frac{a_l^2\bar{\mu}_l u}{(\Delta x_l)^2} + \frac{c}{2\sum_{l=1}^{d}\frac{a_l^2}{(\Delta x_l)^2}}F \tag{17a}$$

$$t^{+} = \frac{c}{2\sum \frac{a_l^2}{(\Delta x_l)^2}} + (1-c)t^{-} + \frac{c}{\sum_{l=1}^{d} \frac{1}{(\Delta x_l)^2}} \sum_{l=1}^{d} \frac{\bar{\mu}_l t}{(\Delta x_l)^2} \qquad 0 < c \leq 1 \qquad (17b)$$

where $\bar{\mu}_l$ is defined by Eq. (9c).

We notice that Eqs. (17a) and (17b) are similar to Eqs. (9a) and (9b). As shall be explained in the Numerical Results section, the value of c which provides maximal time-increment is $c = 1$. In this case, the two sets of equations are identical.

5 Numerical Results

We have implemented and examined our schemes on the multiuser *Sequent Balance* [9] multiprocessor which is a commercial shared bus&shared memory MIMD multiprocessor. The machine we tested on is composed of 20 processors connected by a high speed synchronous bus to a global memory. The processors have no local memory, but do have a local cache.

The following one-dimensional problem is considered:

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} \qquad 0 < x < 1, \quad t > 0$$

with the initial condition,

$$u(x, 0) = \sin(\pi x)$$

and the Dirichlet boundary conditions,

$$u(0, t) = u(1, t) = 0.$$

Its exact solution is given by:

$$u(x, t) = \sin(\pi x) e^{-\pi^2 t}.$$

In addition, we consider the two dimensional problem

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \qquad 0 < x, y < 1, \quad t > 0$$

with the initial condition,

$$u(x, y, 0) = \sin(\pi x)\sin(\pi y)$$

and the following Dirichlet boundary conditions:

$$\begin{aligned} u(0, y, t) &= \sin(\pi y) e^{-2\pi^2 t} \\ u(1, y, t) &= -\sin(\pi y) e^{-2\pi^2 t} \\ u(x, 0, t) &= \sin(\pi x) e^{-2\pi^2 t} \\ u(x, 1, t) &= -\sin(\pi x) e^{-2\pi^2 t} \end{aligned}$$

Its exact solution is given by:

$$u(x, y, t) = \sin(\pi x)\sin(\pi y) e^{-2\pi^2 t}$$

The following parameters are used:

- $\Delta x, \Delta y$ - represent the discretization lengths along the $x-$axis and the $y-$ axis, respectively.
- Δt - represents the discretization length along the $t-$axis.
- T - The time level to be reached for the given problem.
- p - The number of processing elements used for calculation.
- L - The number of total interior grid points in the considered domain.
- e - The maximal error magnitude.
- E - The maximal absolute relative error of the numerical result (relative to the analytic solution).
- *scheme* - Four types of parallel schemes were used: Eq. (2) (SY), Eq. (3) (AS), Eqs. (5a)-(5c) (CA), Eqs. (9a) and (9b) (TS) and SR, the serial version of Eq. (2).
- n_{scheme} - Number of time-steps needed to the named *scheme*, in order to reach $t = T$.

As can be expected from Eq. (9b), run time has been shown to be linearly proportional to c for $0 < c < 1$ and $c = 1$ has provided the fastest runs in all cases. Note that in this case, operations are saved since the expressions are simplified. Furthermore, in conjunction with the assumptions of Lemma 3, for large $n_{\mathbf{j}}$ we obtained in our numerical simulations that

$$\sum_{l=1}^{d} \frac{a_l^2 \delta_l^2 t}{(\Delta x_l)^2} = O(1)$$

In particular, we obtained

$$\Delta t_{\mathbf{j}}^{n_{\mathbf{j}}} \approx \frac{c}{\sum_{l=1}^{d} \frac{a_l^2}{(\Delta x_l)^2}}$$

(see Eq. (8)), and

$$t_{\mathbf{j}}^{n_{\mathbf{j}}} \approx n_{\mathbf{j}} \frac{c}{\sum_{l=1}^{d} \frac{a_l^2}{(\Delta x_l)^2}}$$

This behavior is depicted clearly by Tables 1 and 2 which present the maximal number of time steps within each processor needed to reach the first time $T = 0.1$, in our one-dimensional and two-dimensional problems, respectively. In both cases $c = 1$.

We note that very small variations of n_{TS} occurred within each processor among its own grid points. In either the one-dimensional case or in the two-dimensional case $n_{\mathrm{TS}} \approx \frac{1}{2} n_{\mathrm{SY}}$.

Tables 3 and 4 present the efficiency, in the one and two-dimensional case, respectively, given by:

$$\mathrm{Efficiency} = \frac{\tau_{\mathrm{SR}}(1, \mathrm{L})}{\mathrm{p}\tau_{scheme}(\mathrm{p}, \mathrm{L})}$$

where $\tau_{\mathrm{SR}}(1, L)$ is the run time of the serial version solving with its single processor a problem of L grid points, and $\tau_{scheme}(p, L)$ is the run time of the above parallel schemes, solving the same problem with p processors.

processor ID	0	1	2	3	4	5	6	7	8	9
Max n_{TS}	258	248	246	257	249	246	252	242	243	261

Table 1: Maximal Iteration number for 1-D prob. with 10×5 grid points, $T = 0.1$ ($n_{\mathrm{SY}} = 480$)

processor ID	0	1	2	3	4	5	6	7
Max n_{TS}	57	52	51	53	51	52	53	53

Table 2: Maximal Iteration number for 2-D prob. with $8 \times 2 \times 16$ grid points, $T = 0.1$ ($n_{\mathrm{SY}} = 90$)

p	Efficiency			
	SY	AS	CA	TS
2	0.81	0.99	0.54	1.78
3	0.76	0.97	0.52	1.69
4	0.69	0.94	0.49	1.57
6	0.64	0.89	0.47	1.38
8	0.5	0.75	0.43	1.16
12	0.35	0.56	0.37	0.85
16	0.26	0.43	0.29	0.52

Table 3: Efficiency for the one dimensional problem with p processors and $\frac{L}{p}$ grid points per processor; $L = 48, \Delta x = 0.02128, \frac{\Delta t}{(\Delta x)^2} = 0.5, T = 0.5$ and $c = 1$.

As observed, the efficiency of our time-stabilizing scheme is about the same as that of the asynchronous scheme in certain cases, and it is much higher in other cases. Furthermore, in all cases, the TS scheme efficiency is significantly higher than that of the corrected-asynchronous scheme; in certain cases it is even tripled. This is in spite of the fact that extra interpolation is needed to evaluate the solution at a specific time value, in the TS scheme. It should be noted here that since we are using a multi-user system, the corresponding delays of the serial and the parallel schemes may not be the same. For this reason, the efficiency might be in certain cases slightly greater than one (see the AS scheme in Table 4). However, in the TS case, as explained above, for a given T the number of time-steps is reduced by about 2, as compared to the SY (or SR) case. Therefore, in this case, the efficiency may be significantly better. This is even more evident in cases where the parallel algorithms exploit the cache better than the serial algorithm in which all the data is referenced by the same processor. As a result the efficiency becomes much higher than one (see Table 3).

As was already indicated in [2] and [3] the asynchronous scheme is not consistent with the

p	pts in blk	Efficiency			
		SY	AS	CA	TS
2	8×16	0.92	1.14	0.53	1.09
4	4×16	0.87	0.98	0.47	1.04
8	2×16	0.75	0.87	0.34	0.92
16	1×16	0.36	0.5	0.3	0.63

Table 4: Efficiency for the 2-dim. problem with p processors; $L = 256$, $\Delta x = \Delta y = 0.0667$, $\frac{\Delta t}{(\Delta x)^2} + \frac{\Delta t}{(\Delta y)^2} = 0.5$, $T = 0.5$ and $c = 1$.

		E				e			
p	pts	SY	AS	CA	TS	SY	AS	CA	TS
2	24	0.0036	0.98	0.02	0.05	0.000026	0.007	0.00008	0.0002
3	16	0.0036	0.98	0.05	0.09	0.000026	0.007	0.00022	0.0004
4	12	0.0036	0.98	0.05	0.13	0.000026	0.007	0.00022	0.0006
6	8	0.0036	0.98	0.05	0.09	0.000026	0.007	0.0002	0.0004
8	6	0.0036	0.95	0.05	0.06	0.000026	0.006	0.0002	0.0002
12	4	0.0036	0.96	0.05	0.04	0.000026	0.007	0.0002	0.0006
16	3	0.0036	0.96	0.04	0.02	0.000026	0.007	0.00012	0.00008

Table 5: Accuracy for the one dimensional problem and with p processors and $\frac{L}{p}$ grid points per processor; $L = 48, \Delta x = 0.02128$, $\frac{\Delta t}{(\Delta x)^2} = 0.5, T = 0.5$ and $c = 1$.

heat equation. It is accurate only for steady-state results, while the corrected asynchronous scheme provides accurate results also for intermediate (non steady-state) values. Tables 5 and 6 show the accuracies of our schemes in the one-dimensional and two-dimensional cases, respectively. As noticed, for small number of processors the CA scheme is slightly more accurate than the TS scheme in the one-dimensional case and both provide about the same accuracy in the two-dimensional case. It should be emphasized here that due to the randomness of the phase delays, the shown accuracies of the different approximations were not obtained under the same circumstances. Different runs of the same approximation provided slightly different results and in other cases (not shown here) the TS was slightly more accurate than the CA scheme. However, one can say that in general the CA and the TS schemes provide about the same accuracy. Both the TS and the CA schemes are less accurate than the synchronous scheme, because of the lower order of their truncation error ($O(\Delta x_l)$ instead of $O(\Delta x_l^2)$). Due to the dimensional additivity of the truncation error, these schemes are even less accurate for multidimensional problems. For such problems, if high accuracy is sought, in general one should use higher order schemes [1] since for the TS one would then need a very fine grid that would increase its run-time substantially and make the SY approximation preferable. Nevertheless, under certain circumstances, when the spatial

p	pts in blk	E				e			
		SY	AS	CA	TS	SY	AS	CA	TS
2	8×16	0.07	0.99	0.17	0.5	0.0000036	0.00005	0.000003	0.00002
4	4×16	0.07	0.99	0.5	0.21	0.0000036	0.00005	0.000003	0.000008
8	2×16	0.07	0.97	0.17	0.12	0.0000036	0.000049	0.0000021	0.0000022
16	1×16	0.07	0.99	0.17	0.18	0.0000036	0.00005	0.000003	0.000002

Table 6: Accuracy for the two-dimensional problem with p processors; $L = 256$, $\Delta x = \Delta y = 0.0667$, $\frac{\Delta t}{(\Delta x)^2} + \frac{\Delta t}{(\Delta y)^2} = 0.5$, $T = 0.5$ and $c = 1$.

	SY			TS			$\frac{\tau_{TS}}{\tau_{SY}}$
p	$\frac{L}{p}$	Eff	E	$\frac{L}{p}$	Eff	E	
2	500	0.68	0.18209×10^{-14}	550	0.80	0.16251×10^{-14}	0.86
4	250	0.67	0.18209×10^{-14}	275	0.79	0.15797×10^{-14}	0.86
5	200	0.67	0.18209×10^{-14}	220	0.77	0.08089×10^{-14}	0.86
8	125	0.61	0.18209×10^{-14}	137	0.74	0.02522×10^{-14}	0.84
10	100	0.59	0.18209×10^{-14}	110	0.70	0.01740×10^{-14}	0.85

Table 7: Comparison of efficiency (Eff) and accuracy (E) of TS and SY for a 1-D Dirichlet problem in the interval $20 \leq x \leq 820$ with p processors and $\frac{L}{p}$ grid points per processor; $L = 1000, \Delta x = 0.8, \frac{\Delta t}{(\Delta x)^2} = 0.5$, and $n = 312$ for SY while $L = 1100, \Delta x = 0.7, c = 1$ and $T = 100$ for TS.

increments are relatively large, the accuracy of the TS may become about the same as that of the SY scheme (which, in turn, is the same accuracy as of the SR approximation) and yet providing higher efficiency. This is illustrated by Table 7 which compares the performance of SY and TS for the following problem:

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} \qquad 20 \leq x \leq 820$$

$$u(x,0) = \frac{1}{2}x^2$$

$$u(20,t) = 200 + t \quad , \quad u(820,t) = 336.2 \cdot 10^3 + t \quad .$$

The analytic solution is

$$u_A(x,t) = \frac{1}{2}x^2 + t \quad .$$

The spatial increments are $\Delta x = 0.8$ for SY, and $\Delta x = 0.7$ for TS. For both cases the time increment is $\Delta t = 0.5(\Delta x)^2$. Each processor is assigned $\frac{L}{p}$ grid points with $L = 1000$ for SY and $L = 1100$ for TS. We observe that the TS method provides about the same and sometimes even better accuracy and yet its efficiency is more than 15% higher as compared to the SY method.

6 Summary

This paper presents new asynchronous and *Time-Stabilizing* (TS) finite-difference schemes for constant-coefficient parabolic PDEs. These schemes remove the synchronization overhead and are suitable for message-passing, shared-memory, single- or multi-user multiprocessors. They generalize the CA scheme developed in [2] and [3] by allowing non-constant time increments. Our schemes are analyzed and implemented on the multiuser shared-memory Sequent Balance multiprocessor. Their performances are compared with the performances of the CA scheme, and of the corresponding serial (SR), synchronous (SY) and asynchronous (AS) schemes.

It is shown that in the TS scheme the derivatives of the grid-points time-level with respect to the spatial coordinates are bounded as the time-step number increases, and they vanish as the spatial mesh-sizes approach zero. It is for this reason that unlike previous schemes, the TS scheme can be implemented on multiprocessors with processors of different speeds/workloads, as long as these differences are not too large.

The efficiency of the TS scheme is about the same as of the AS scheme in certain cases and much higher in other cases. In all cases the TS scheme is much more efficient than the CA scheme. The AS scheme provides accurate results only for steady-state values [3]. As for intermediate (non steady-state) values, the accuracy of the TS approximation is about the same as of the CA approximation. Both are less accurate than the SY scheme, because of the lower order of the truncation error ($O(\Delta x_l)$ instead of $O(\Delta x_l)^2$). Due to the dimensional additivity of the truncation error, these schemes are less accurate especially for multidimensional problems. For such problems, if high accuracy is sought, one should use higher order schemes. Nevertheless, under certain circumstances, when the spatial increments are relatively large, the performance (i.e both efficiency and accuracy) of TS may still be better.

References

[1] D. Amitai, A. Averbuch, M. Israeli and S. Itzikowitz, General Parallel Finite-Difference Schemes for Constant-Coefficient Parabolic PDEs, *ICIAM Conference*, June 8-12, Washington DC., USA (1991).

[2] D. Amitai, A. Averbuch, S. Itzikowitz and E. Turkel, Asynchronous and Corrected-Asynchronous Numerical Solutions of Parabolic PDEs on Parallel MIMD Multiprocessors, *submitted for publication.*

[3] D. Amitai, A. Averbuch, S. Itzikowitz and E. Turkel, Asynchronous Numerical Solution of PDEs on Parallel computers, *Proc. of the 4^{th} SIAM Conference on Parallel Processing for Scientific Computing*, 131-136, Dec. 11-13, Chicago, USA (1989).

[4] G. M. Baudet, Asynchronous Iterative Methods for Multiprocessors, *J. of the ACM*, **25**, 226-244 (1978).

[5] D. Chazan and W. Miranker, Chaotic Relaxations, *J. of Linear Algebra Applications*, **2**, 199-222 (1969).

[6] D.J. Evans, New parallel algorithms for partial differential equations, *Parallel Computing 83* (Eds J. Joubert M. feilmeier and U. Schendel), 3-56. Elsevier Science Publishers B.V., North Holland (1984).

[7] D. Mitra, Asynchronous Relaxations for the Numerical Solution of Differential Equations by Parallel Processors, *SIAM J. Sci. Stat. Comput.*, **8**, 43-53 (1987).

[8] J. M. Ortega and R. G. Voigt, Solution of Parallel Differential Equations on Vector and Parallel Computers, *SIAM Review*, **27**(2), 149-239 (1985).

[9] Sequent Systems, The Sequent Guide to Parallel Programming (1987).

[10] V.K. Saulyev, *Integrating Equations of Parabolic Type by the Methods of Nets*, Pergamon Press, Elmsford, New York(1964).